**The Report Committee for Prateek Burman**
**Certifies that this is the approved version of the following report:**

# Quadcopter stabilization with Neural Network

**APPROVED BY**
**SUPERVISING COMMITTEE:**

**Supervisor:**

Christine Julien

William (Bill) Bard

# Quadcopter stabilization with Neural Network

**by**

**Prateek Burman, B.S.E.E**

**Report**

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

**Master of Science in Engineering**

**The University of Texas at Austin**

**December 2016**

# Dedication

This report is dedicated to my parents, my sister and memory of my grandparents.

# Acknowledgements

I would like to extend my utmost sincere gratitude to Dr. Christine Julien, without her guidance I would have strayed away from the objective not been able to finish the project.

I also express my gratitude to Prof. William (Bill) Bard for being the reader for my report. Prof. Bard's class on Advanced Embedded Systems was the inspiration for this report.

Finally, I would also like to thank my parents, my sister Priyanka Burman and Anjana Parandhaman for believing in me and for all the support.

# Abstract

## Quadcopter stabilization with Neural Network

Prateek Burman, M.S.E

The University of Texas at Austin, 2016

Supervisor: Christine Julien

UAVs (Unmanned Aerial Vehicle), also known as drones, are becoming attractive in the consumer space due to their relatively low cost and their ability to operate autonomously with minimal human intervention. A user could program the drone with GPS coordinates, and the drone would comply with utmost precision. In order for the drone to operate a preprogrammed flight path, it requires a host of sensors for it to gather data and operate on that data in real time. For instance, a consumer drone typically has obstacle avoidance sensors, a GPS sensor for routing and navigation, and an IMU (Inertial Measurement Unit) for tracking position and orientation. These sensors play a crucial role in both stabilization and navigation of the drone. This report aims to investigate, analyze and understand the complexity involved in designing and implementing an autonomous quadcopter; specifically, the stabilization algorithms. In general, stabilization is achieved using some form of control algorithm. The report covers a popular approach for stabilization (PID Control) found with many open source libraries and contrasts it with an alternative machine learning approach (Neural Networks). Finally, a machine learning

based algorithm is implemented and evaluated on a prototype quadcopter, and its results are presented.

# Table of Contents

# List of Figures

# List of Tables

# INTRODUCTION

The application of unmanned aerial vehicles (UAV) has been gaining popularity due to the ease with which the aerial vehicle (quadcopters for instance) can perform tasks such as obstacle avoidance without collision, vertical take-off and landing (VTOL), and maneuvering without any direct human involvement. If a quadcopter is operating in an indoor environment, it needs to do so with agility and snap responses without going out of control. In an outdoor environment, the drone needs to stabilize itself against external forces. For both scenarios, drone's ability to operate a space stable flight is extremely important to their functionality; it can also prevent the quadcopter from crashing in the event of heavy external disturbance such as a strong wind.

All of these applications require drones to be equipped with a host of sensors like an accelerometer, magnetometer, gyroscope, and depth-camera to allow UAVs to perform stabilization and navigation.

Multirotor UAVs tend to be unstable mainly due to two reasons; the first being, the sum of the moments generated by all the rotors has to equal zero during the hover state, to avoid aerial device from spinning about its access. The second reason being not all rotors will respond exactly to the same input. Hence, a control algorithm is used to constantly adjust the output based on the current state as described by the sensors.

The onboard processor uses the fused data from the sensors to build a control system using a Proportional-Integral-Derivative (PID) controller. This is a closed loop controller, which works by applying gains to an error signal to steer the UAV towards the desired goal location. The simplicity of PID control makes it a popular choice for implementing in multirotor UAVs. PID control algorithm requires parameter tuning; engineers generally use trial and error for determining parameter values. This process is

time consuming and prone to errors. The alternative to manual tuning is to use Ziegler-Nichols rule for tuning; however, even this method is not ideal, as it yields aggressive gains and overshoot. Ziegler-Nichols method is not very robust and can also lead to instability. Since we want a stable hovering and navigation for our multirotor, we have to be able to minimize overshoot without dampening the system too much. A well designed and implemented hover stabilization algorithm will allow the quadcopter to hover at fixed altitude, regardless of the changes introduced by external forces.

Hence, a Neural Network (NN) based approach for quadcopter stabilization is presented as an alternative, which offers a more robust stabilization over PID controller to unseen inputs and noisy sensor data. The NN is trained with PID control system and the learnt weights are applied to the quadcopter during the test flight. Finally, the results are contrasted against the PID controller output.

# LITERATURE REVIEW

Quadcopters are non-holonomic underactuated systems with six degrees of freedom (translation along x, y, z axis and rotation around x, y, z axis) and only four rotors to make use of the controllable degrees of freedom (roll, pitch, yaw, altitude). These systems are also dynamically unstable and nonlinear. The quadcopter algorithm structure is typically implemented as two control algorithms in a cascade arrangement, where the inner-loop controls the attitude and the outer-loop performs position control. The cascade architecture allows quadcopter to address multiple disturbances and recover immediately. Different control methods can be used to implement the cascade architecture such as, PID control, H control, predictive control, sliding mode control, neural network control. Despite the advancements in control theory, traditional and most common approach to quadcopter stabilization remains PID control [1] or some combination of PID such as PD controller [2], mainly due to its simplicity and effectiveness.

The PID controllers work by acting on each of the quadcopter states: roll, pitch, yaw, altitude and position. However, PID controllers require tuning and setting of Proportional, Integral and Derivative multipliers. The process of optimizing these controllers is a time-consuming and laborious process with limited success (stability), if tuned incorrectly. The authors of [3] have used Ziegler-Nichols Rules to determine PD parameters for their quadcopter control, however, the system is still prone to high overshoot values. Regardless, PID controllers in general have trouble operating in conditions that are too far from the optimal hover point, especially in the case of micro aerial vehicles as disturbances due to wind can be large [4].

Therefore, neural network based solutions have been an attractive alternative [6, 7, 8], as they can provide a higher level of robustness and adaptability with equal or less effort

on training. Neural networks create a non-linear mapping from inputs to outputs that can capture quadrotor dynamics to create a robust controller.

The amount of literature for neural network based control is vast and wide ranging. For instance, the authors of [7] propose an adaptive neural network control based on mammalian cerebellum called CMAC (Cerebellar Model Articulation Controller) for stabilization especially during considerable wind disturbance. However, CMAC algorithm can require huge amounts of memory, hence those may not be feasible to implement on an embedded system. Researchers have also proposed self-tuning adaptive control using neural networks in [8].

Another approach includes combining genetic algorithm with neural networks to create neuro-evolutionary algorithms, which ranks each controller based on some cost function [9]. Higher performing cost functions are mutated, with some probability, in order to search for optimal solution. This approach as demonstrated by Shepherd and Tumer in [7] is impressive as the system should be capable of designing an optimum solution on its own. However, in practice these are hard to implement and debug. Also, to reach an optimum solution could take many generations.

On the other hand, Lower and Wroclaw contrast a classic PID controller with a neural network based controller in [10]. The neural network is taught by control system with a standard PD controller. The simulation results of the neural controller and PID controller are then compared. A similar setup is done by the authors of [11] for a hexacopter. Both of these approaches are much simpler, yet can yield effective results.

The motivation behind the paper is to showcase an alternative approach which can overcome the of shortcomings of the PID control algorithm. A PID control algorithm is susceptible to large oscillations and disturbances, causing the system to go out of balance easily and not recover from such states. In this paper a neural network based approach is

presented for quadcopter stabilization. Finally, the output of the NN based controller is contrasted against the PID controller.

# APPROACH

## Sensors

### INERTIAL MEASUREMENT UNIT (IMU)

The quadcopter uses a Sensor Hub BoosterPack containing a microelectromechanical (MEMs) sensor which internally has a 3-axis gyroscope, a 3-axis accelerometer and a 3-axis digital compass; thus allowing 9 degrees of motion tracking.

As discussed later in the paper that the accelerometer yields noisy data and the gyroscope is prone to drifting. Hence, there is a need for data fusion to generate a more consistent, accurate, and reliable representation of quadcopter's attitude from disparate sensors, which are less accurate individually.

| | X | Y | Z |
|---|---|---|---|
| Accel | -4.960 | 0.803 | 8.470 |
| Gyro | 0.010 | 0.013 | 0.015 |
| Mag | 2.700 | 37.500 | -9.300 |

Figure 1: Example of output from an IMU measurement.

### BAROMETER

The Sensor Hub BoosterPack also includes a BMP180 pressure sensor. Since we know pressure changes with altitude, I used BMP180 as an altimeter. By applying the following formula, I was able to obtain the altitude (in meters), where P0 is the average pressure at sea level (101325 Pa) and P is the pressure obtained from the sensor. The altitude of the quadcopter is used to evaluate its current state for hovering. By monitoring

the rate of change in altitude, I was also able to evaluate the stability of the quadcopter for a given environment.

$$altitude = 44330 * \left(1 - \left(\frac{p}{p_o}\right)^{\frac{1}{5.255}}\right)$$

Equation 1: Calculating altitude from temperature and pressure.

## Sensor fusion

The MEMs based gyroscopes are lightweight and have low power consumption, however, they also have the disadvantage of having a high level of measurement noise and bias. Over time the measurement will drift, mainly due to accumulation of integration error and it will not return to zero, when the system is back to its original position. Hence, the gyroscopic data is only reliable only in the short term.

Similarly, the accelerometer measures forces acting on the body; these forces are not limited to the ones applied intentionally, but can be gravity, vibrations at high frequencies or external disturbances (such as wind). The accelerometer data can be very volatile in the short term, hence, a low pass filter can be used to only look at reliable long term data.

Accelerometers can only be used to measure tilt and not heading; therefore, I filtered out the low frequency gyroscopic bias and high frequency accelerometer vibration when combining the data together. Typically, an algorithm such as Kalman filter is used to fuse the data from multiple sensors in order to create a more accurate estimation of the quadcopter orientation in space [10].

In my case I fused 9 axis measurements into a set of Euler angles (roll, pitch, yaw). The fusion mechanism used is complementary-filtered direction cosine matrix (DCM) algorithm, provided as part of the Sensor library. A complementary-filter is computationally more efficient and is simpler to understand compared to Kalman filter.

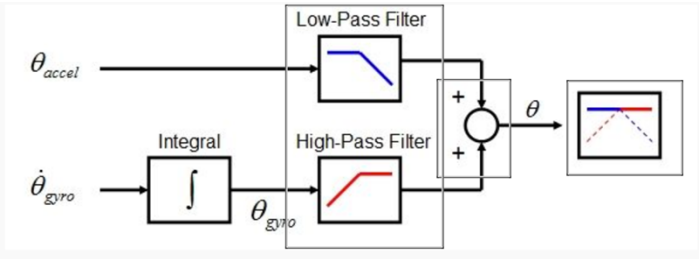


Figure 2: DCM visual explanation.

I used a complementary-filter to get data from gyroscope in the short term, as it is not susceptible to external forces like an accelerometer. For the long term I used accelerometer data, as it does not drift over time.

The filtered readings from the sensor are fed to the DCM algorithm to calculate the orientation of a rigid body relative to the earth's magnetic field and direction of gravity expressed in Euler angles (roll, pitch and yaw).

Finally, the results from the DCM algorithm determine the current state of the system. These values are fed into the controller for stabilization and navigation.

## Flight dynamics

A quadcopter is a four rotor multicopter capable of altering its position and orientation in 3D space by altering its moment on the rotor. The change in moment is achieved by altering the power supplied to the rotor (using Pulse Width Modulation

technique) to change the engine speed in order to generate a different balance of forces; thereby altering the orientation of the quadcopter. All the rotors contribute to the upward thrust, however, they also contribute towards rotational torque. Hence, half of the rotors are programmed to spin clockwise (CW) and the other half to spin counter-clockwise (CCW). In my case, I configured the quadcopter in a plus (+) configuration where the front and the back rotors spin in CW direction, while the left and right spin in CCW motion; thereby having a net torque of zero during hover configuration.

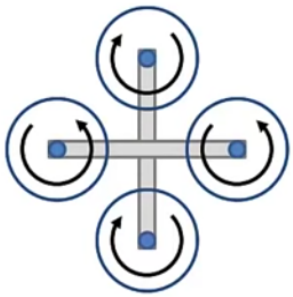

Figure 3: Sample quadcopter view from the top.

The controller algorithm determines the current state of the quadcopter based on the sensor readings and compares against a desired set-point. It is then, the controller will evaluate a change of state to achieve the desired state. The main objective of the controller is to achieve the set-point as quickly as possible without overshooting the target or causing the system to go out of balance.

**Altitude change:**

To change the altitude of the quadcopter, I either increase or decrease the speed equally among all four motors; thereby moving the quadcopter along the vertical axis without changing the orientation of the quadcopter.

**Yaw change in hover state:**

To change the yaw (the angle along the z axis) without changing altitude I increase the speed of two opposite rotors, while decreasing the speed on complementary pair of rotors by the same proportion. In order to yaw clockwise, I increase the speed on front and back rotors while subtracting the same offset from counter-clockwise pair. To spin CCW (when viewed from top) a desired CCW yaw would have an offset subtracted from both CW motors and the same offset added to both CCW motors, in order to maintain altitude.

**Pitch change in hover state:**

To alter the pitch (the angle along the y axis) without altering the altitude I increase the speed on one of the rotors on the y axis while decreasing the speed by the same offset on the complimentary rotor on the y axis. The motor speed for both motors on the y axis must be changed in order to pitch without changing the altitude; thus changing pitch; but not changing the overall throttle of the system.

**Roll change in hover state:**

Rolling the quadcopter is similar to pitching in hover configuration; I increase the speed of one of the motors along x axis, while decreasing the speed by the same offset on the complementary motor on x axis. Again, I maintain the overall throttle of the system, as I increased the speed on one motor by the same value as that decreased on the other motor; thereby, maintaining a hover state.

# Control algorithm

**PID CONTROLLER:**

Control algorithms monitor and alter the operating conditions of a dynamic system, with the goal of bringing the system to a desired state. The controller evaluates the error e(t) in the system from the difference between the setpoint w(t) and the measured output of the system y(t). The system then applies a change in the system to drive the error towards zero. A Proportional-Integral-Derivative controller (PID controller) is a closed control loop applying gains to an error signal and feeding the result back as input to the plant. It is most commonly used in the industry mainly due to its simplicity, effectiveness and it can be tuned even when plant is not available [12].

The error value is calculated as the difference between the desired setpoint and measured value. The controller works by minimizing the error over time by adjusting the power to the rotors. There are 3 types of gain which are calculated on every run.

The proportional gain is directly proportional to the error; it is the main driver of the system and works directly to bring the system to the target setpoint. The larger the proportional gain, the faster the system responds to the error. However, this can lead to overshooting the target setpoint and lead to oscillation and possibility of an unstable system. The integral gain is the running sum of the error over a period. This term attempts to account for the steady state error in the system. Finally, the derivative gain is the change in error. This term accounts for the rate of change of error and prevents the system from overshooting its target. Each of the gains is amplified with a coefficient multiplier and together the sum of each of the gains is what drives the change in valve (power to the rotor).

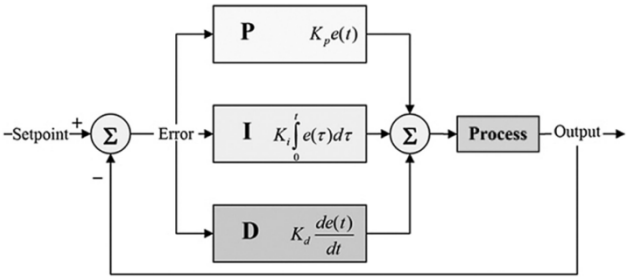


Figure 4: A block diagram of a PID controller.

$$u(t) = K_p * e(t) + K_i * dt \int_0^t e(t) + K_d * \frac{de(t)}{dt}$$

Equation 2: PID equation.

PID controller require tuning of the three coefficients of the model to meet the needs of the underlying system. PID controller is evaluated in terms of its responsiveness, the overshoot and the system oscillation. Each of these features can be altered by picking the $k_p$, $k_d$ and $k_i$ terms. Typically, empirical method such as trial and error is used to derive the coefficient values. This of course can be a tedious and a time consuming process and do not always yield optimal results. Later in the report I evaluate an alternative approach for quadcopter stabilization.

In my design I used a PD controller to steer the quadcopter towards a desired goal altitude and attitude. I use the altitude, position and velocity estimates from DCM and I apply a PD control to steer it towards a desired goal. The quadcopter has four controllable degrees of freedom (altitude, roll, pitch, yaw); therefore, I employ a separate PD control per dimension to drive the quadcopter to stability. The PD coefficients were picked using trial and error by monitoring the stability of the quadcopter over a 10 second test flight.

**NEURAL NETWORK CONTROLLER:**

The alternative approach to PID controller proposed in this paper is using Neural networks (NN). NN is a computation approach representing a generalized mathematical model of the human neuron collection and inter-connection between them. NN aims to provide the same approach as a human brain to solve a given problem. Since, NN can non-linearly map inputs to the outputs, they can be used to replace our PD controller.

Each neural node sums the incoming signal and generates an output signal based on a predefined function, also known as activation function. The activation function allows the network to generate non-linearity between the inputs and outputs. In our case the predefined function is a sigmoid function. Each neuron is connected by weight with many others to form input of an adjacent neuron of the next layer, thereby forming a network of neurons. This processing of input through the hidden layer to the output nodes is called feedforward network.

$$S(x) = \frac{1}{1 + e^{-x}}$$

Equation 3: Sigmoid function.

A NN learns from the training samples, thereby, updating the weight associated with each of the neural nodes. With every training sample each of the weights are updated by either inhibiting or enforcing the incoming signal based on the result.

In order for the NN to learn and compare system output with the desired output a technique called backpropagation is used. It is one of the more common techniques used to (learn) update the weights of the network, that is, train our network. With backpropagation, error (desired output is subtracted from calculated output) is propagated

backwards from the output towards the input; and the weights are updated in the process to reflect the error.

Once the network learns the pattern between the input and the output, it is then used as a replacement for the PID controller to evaluate the new inputs in real-time.

# IMPLEMENTATION

## High Level Design

The frame of the quadcopter is built from Crazy2Fly Hardware kit [13]. The design of the quadcopter uses four 12 Amp Electronic Speed Controller (ESC) to drive four respective 1000kv brushless motors for generating thrust. The ARM Cortex based TM4C123G microcontroller is the brain controlling the brushless motors by providing the appropriate signal using pulse width modulation (PWM) technique to the ESC. The duration of the high (on) signal period over the low signal (off) period using PWM determines the power supplied to the load, this is also known as duty cycle. Higher duty cycle results in more power to the motors, which is converted to motor rotations and thereby, thrust (by the propellers) for the system. By controlling the rotations per minute (RPM), the thrust can be controlled for individual motors; and with varying PWM signal, a quadcopter can be maneuvered in one of the six degrees of freedom available to it.

Figure 5: A crazy2fly quadcopter frame.

The microcontroller unit collects sensor readings from the Sensor Hub BoosterPack over I2C port. The Sensor Hub has BMP180 barometer which provides the current

temperature and pressure of the environment. Altitude is then calculated from pressure and temperature using the formula [10] as described earlier in the background section. The Sensor Hub also has MPU9150 which provides acceleration and orientation information; this information is fused using the DCM algorithm (described earlier) to obtain the Euler angles roll, pitch and yaw values.

The control system algorithm is implemented in the controller module of the embedded system using a PD controller. The PD algorithm compares the output (roll, pitch, or yaw values) from the DCM algorithm with the set point to calculate an error value e(t). The algorithm then applies a correction based on the kp, ki and kd values. These corrections are calculated per controllable degrees of freedom (roll, pitch, yaw and altitude) and are called offset. These roll, pitch and yaw offsets are then scaled to appropriate duty cycle and applied to the motor.

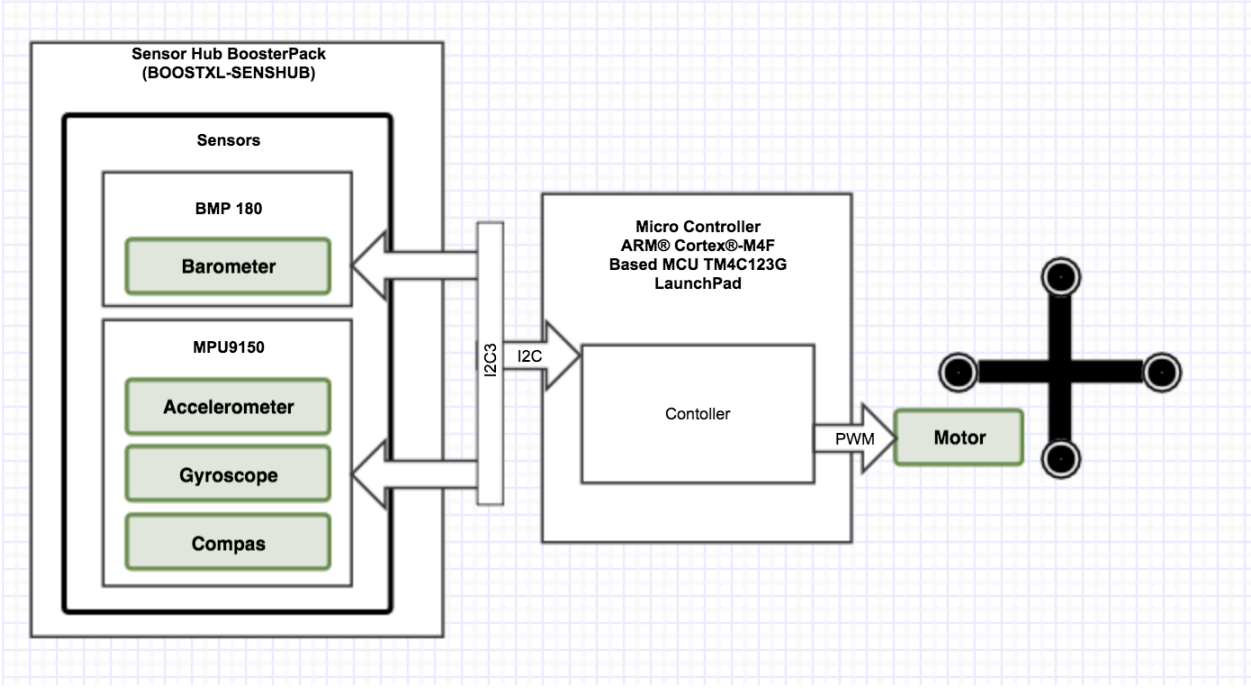


Figure 6: Overall architecture of the system.

```
motor_1Power = map(throttle + pitch_offset + yaw_ffset);
motor_2Power = map(throttle + roll_offset  - yaw_ffset);
motor_3Power = map(throttle - pitch_offset + yaw_ffset);
motor_4Power = map(throttle - roll_offset  - yaw_offset);
```

Equation 4: The general throttle equation for a quadcopter.

One of the disadvantages of PID controllers is that they are not very robust against external disturbances. Thus if a system will encounter challenges from the environment, i.e. a big set point and measure point difference, then PID may not be the ideal approach. The next section describes how the system was changed to use Neural Networks as the controller.

## Neural network controller

For the second approach we replace PD controller with a NN controller using back propagation. The same inputs from the sensor as the PD controller is sent to the NN directly. The output generated by the NN is sent to the PWM module for the motors. The number of hidden layer and neurons per hidden layer was determined experimentally, as too many would cause overfitting to the training input and too few will not map the entire solutions space.

Data for the training is generated from the flight when the quadcopter was controlled using the PD controller. This process of offline tuning is called general training of the NN. The general training can be a CPU intensive process, depending on the size of NN and the volume of training samples. Due to the constraint of limited processing power on embedded systems, this process was implemented on a laptop. Once the network is trained (i.e. the weights optimized), the weights matrix can be copied over to the inflight

controller on the microcontroller, where it'll become a simple feed forward network to the inputs; thereby, replacing the PD controller utilized earlier.

$$(roll, pitch, yaw) \rightarrow NN \rightarrow (m_1, m_2, m_3, m_4); where\ m_1 - m_4 are\ motor\ outputs$$

Equation 5: NN transformation.

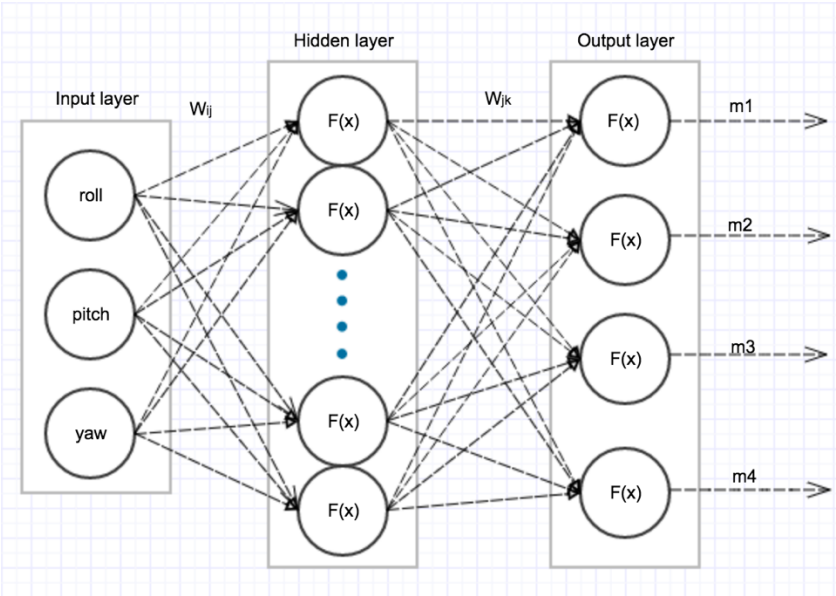


Figure 7: Architecture of the implemented Neural Network.

The raw, pitch, yaw inputs and corresponding controller outputs from the PD controller is collected and split into two set, 80% for training and 20 % for the testing. A normalized set of roll, pitch, yaw values are fed into the input layer; multiplied initially with random weights and the sigmoid activation is applied.

$$F(x) = S(w_{11}x_1 + w_{12}x_2 + \cdots)$$

Equation 6: Function of a Neuron node.

The same step is repeated for between the hidden and the output layer. Finally, the error is calculated between the expected and the calculated values. A training set is used to train the NN by propagating the error backwards from the output towards the input; modifying weights such that the error is minimized. The output values from the neural network are the normalized value for the motors; which are then scaled and fed directly to the PWM module to control the motors.

```
# Create a NN with input layer of 3 neurons;
# hidden layer of 4 neurons and output layer of 4 neurons
hiddenLayer = NNLayer(7, 3)
outputLayer = NNLayer(4, 7)

# Combine the layers to create a neural network
neural_network = NeuralNetwork(hiddenLayer, outputLayer)

# normalize the input
normalized_training_in = drone_training_rpy_in /360
# normalizing by scaling it down to 0 and then dividing by max value.
normalized_training_out = (drone_training_motor_out - 430)/200

data = concatenate((normalized_training_in, normalized_training_out), axis=1)
training_idx = np.random.randint(data.shape[0], size=85)
test_idx = np.random.randint(data.shape[0], size=17)
training, test = data[training_idx, :], data[test_idx, :]

# train the network
neural_network.train(training[:,:3], training[:,3:], 70000)
# Test the neural network
hidden_state, output = neural_network.activation_function(test[:,:3])

# need to scale the output to match the motor's PWM requirement...
scaled_output = neural_network.scale_output(output)

rmse = np.sqrt( np.sum((test[:,3:] - output) ** 2) / test[:,3:].shape[0])
```

Figure 8: Calculating the RMSE for NN controller.

# RESULTS AND OBSERVATIONS

In order to validate the claim made in this paper, this section the two control methods are contrasted. There were two experiments performed. The first being the implementation of PD controller for a brief test flight of 10 seconds.

The first experiment itself required manual tuning of proportional and differential coefficients before a stable flight could be established. Once the $k_p$ and $k_d$ offsets which seem to stabilize the quadcopter are determined, the motor outputs values are collected for a given roll, pitch and yaw values as training data for the NN.

The stability data collected was split in two, training and test set in ratio of 80 to 20 respectively. The training set is then used for the second experiment where a NN based controller was trained offline. The training dataset is used to calculate the weights of the connections between neurons in the network. The strength of the connection determines to either amplify or dampen the incoming signal to generate a desired output. The final weight determined from the training set are listed in Table 1 and Table 2.

The final NN design had seven neurons in the hidden layer, four in the output representing the four motors of the quadcopter and three neurons in the input layer for roll, pitch and yaw.

| | | | | | | |
|---|---|---|---|---|---|---|
| -7.091 | 15.993 | 1.945 | 2.158 | -14.016 | -1.860 | -0.053 |
| -2.258 | 15.664 | -42.185 | -17.561 | -4.011 | 1.592 | -7.793 |
| -11.474 | -5.076 | -15.877 | -1.275 | -2.368 | -3.425 | -18.953 |

Table 1: Input layer weights for NN.

| | | | |
|---|---|---|---|
| 4.612 | 6.288 | -4.612 | -6.288 |
| -1.724 | -1.471 | 1.724 | 1.471 |
| 0.365 | -2.125 | -0.365 | 2.125 |
| 1.902 | -0.830 | -1.902 | 0.830 |
| 0.238 | 3.385 | -0.238 | -3.385 |
| -3.827 | -1.334 | 3.827 | 1.334 |
| -4.188 | -5.432 | 4.188 | 5.432 |

Table 2: Hidden layer weights for NN.

The output of the PD controller ($Y_i$ – expected value) is compared with the output of the NN ($\mathbb{y}_i$ – actual value) for the same input in the test set and the difference is captured as root-mean-square-error (RMSE). It is important to mention a zero value for RMSE represents no difference between PD controller and a NN controller.

$$RMSE = \sqrt{\left(\frac{1}{n}\right) * \sum_{i=1}^{n} (Y_i - \mathbb{y}_i)^2}$$

Equation 7: RMSE equation.

The number of neurons in the hidden layer in NN had a direct impact on the RMSE of the output for the motors. The number of neurons in the hidden layer was determined experimentally, by selecting the number of neurons that minimizes the RMSE. The RMSE value for the test set was 0.0837; which tells us that NN is able to mimic the PD controller very accurately.

As we can see from the experiment the quadcopter that both PD and Neural Network are good at stabilizing. The NN is able to mimic the behavior of the PD controller.

This is mainly because we used PD controller to train the NN. However, in the scenario where the external disturbance is high, the NN based quadcopter should perform better.

# CONCLUSION

In this paper I proposed, built and analyzed the behavior of a quadcopter with two different types of controller. The first controller being the traditional and popular approach of using PID controller. Given the popularity of the PID controller in the industry, it is no doubt a simple and default approach for most control problems. During testing I found PID controller to be better at removing steady state error; however, it would more frequently lead to an unstable quadcopter system. The instability was mainly due to the build-up from the Integral part of the PID controller. If the error between the set point and the measured point was ever large, the quadcopter controller was not able to recover quickly, and the error would persist in from of Integral error for the time period; thus leading to eventual crash. In order to simplify the controller, I decided to use PD controller.

However, the second approach was using neural network controller for stabilization which can achieve set point quicker with lower oscillation (instability) than a PID controller. I was unable to deploy the NN controller on the micro-controller to see whether a NN based quadcopter is better able to cope with input noise and external disturbances while improving on stability. This is something I would like to continue to validate in future. We know from prior research [10] that NN based controller can be more stable to measurement and external noise than a PID based controller.

In this paper, we did see that NN performs very similar to PD controller, since NN is trained with PD controller output. Hence, NN despite being more complicated than PID based controller, they give us the flexibility to add more robustness later down the road. For example, we could expand the neural network to include altitude sensor values or include the rate of change of roll, pitch and yaw values. The other added advantage of NN based controller is that we can use a single controller for multiple dimensions (roll, pitch,

yaw) and completely get rid of Equation 4. With NN controller we can calculate all four motor values, instead of having a separate controller per dimension in case of our first approach with PID controller and then formulating them into motor outputs.

A PID controller based implementation, also limits user from making improvement after the initial selection of the coefficients. One can only make it better by selecting a better set of coefficients, which again can be tiring process.

In the future the ideal next step would be to expand on the neural network work and instead of training the network using PD control, an unsupervised approach for a self-tuning quadcopter stabilization can be utilized such as recurrent neural networks. This approach would allow NN to become completely independent from PID based tuning and possibly allow us to better stabilize the quadcopter. Even though NN are more process intensive compared to PID controller; the availability of faster and resource efficient processors on embedded devices these day, it is completely possible to train the NN on the fly.

# Bibliography


[1] Cowling, I. Yakimenko, O. Whidborne, J. and Cooke, A. A prototype of an autonomous controller for a quadrotor UAV. European Control Conference, 2007.

[2] Erginer, B. and Altug, E. Modeling and PD control of a quadrotor VTOL vehicle. IEEE Intelligent Vehicles Symposium, pages 894–899, Jan 2007.

[3] He, Z. Zhao, L. A Simple Attitude Control of Quadrotor Helicopter Based on Ziegler-Nichols Rules for Tuning PD Parameters. The Scientific World Journal, 2014.

[4] Shepherd III, J. F. and Tumer, K. Robust neuro-control for a micro quadrotor. In Proceedings of the 12th annual conference on Genetic and evolutionary computation, GECCO '10, pages 1131–1138, 2010.

[5] Das, A. Lewis, F and Subbarao, and K. Backstepping approach for controlling a quadrotor using lagrange form dynamics. Journal of Intelligent and Robotic Systems, pages 127–151, 2009.

[6] Madani, T. and Benallegue, A. Adaptive control via backstepping technique and neural networks of a quadrotor helicopter. Proceedings of the 17th World Congress of The International Federation of Automatic Control, 2008.

[7] Nicol, C. Macnab, C. and Ramirez-Serrano, A. Robust neural network control of a quadrotor helicopter. Canadian Conference on Electrical and Computer Engineering, Jan 2008.

[8] Fu-Chuang, C. Back-Propagation Neural Networks for Non-linear Self-Tuning Adaptive Control. IEEE Control Systems Magazine, Volume 10, Issue 3. Apr 1990.

[9] Stanley, K. O. and Miikkulainen, R. Evolving neural networks through augmenting topologies. Evolutionary Computation, pages 99–127, 2002.

[10] Lower, M. and Tarnawski, W. Quadrotor Navigation Using the PID and Neural Network Controller. University of Technology, Wybrzeże Wyspiańskiego 27, pages 50-370 Wrocław, Poland.

[11] Artale, V. Collotte, M. Milazzo, C. Pau, G. and Ricciardello, A. Real-Time System based on a Neural Network and PID Flight Control. Applied Mathematics & Information Sciences. Mar 2016.

[12] Ang, K. H. Chong, G. Yun, L. PID control system analysis, design, and technology. IEEE Transactions on Control Systems Technology, pages 559- 576. July 2005.

[13] "Crazy2Fly," [Online] Available: http://www.lynxmotion.com/p-901-crazy2fly-quadcopter-base-combo-kit.aspx.